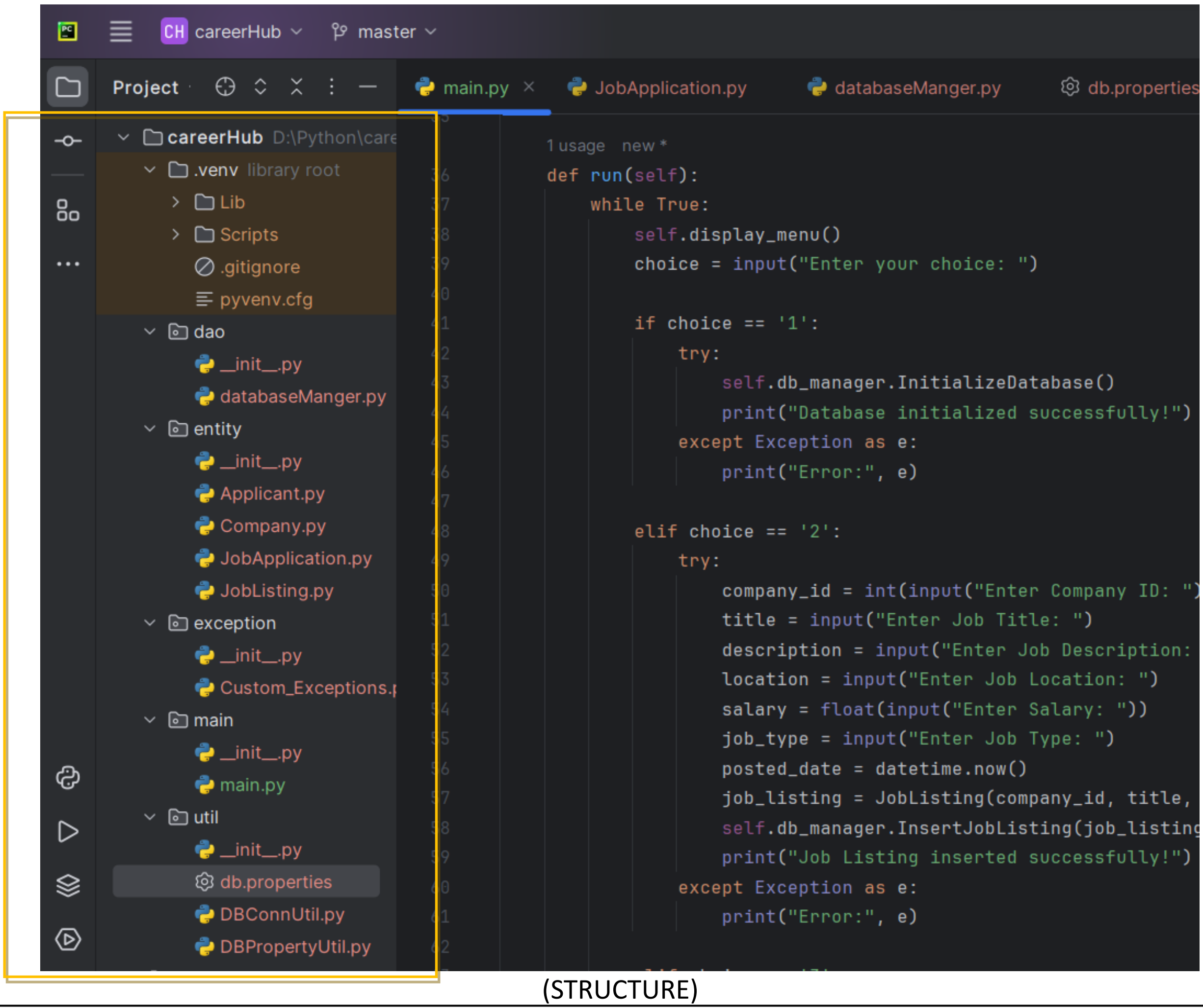
- The following Directory structure is to be followed in the application.
  - **entity/model**
    - Create entity classes in this package. All entity class should not have any business logic.
  - **dao**
    - Create Service Provider Interface/Absract Class to showcase functionalities.
    - Create the implementation class for the above Interface/Absract Class with db interaction.
  - **exception**
    - Create user defined exceptions in this package and handle exceptions whenever needed.
  - **util**
    - Create a DBPropertyUtil class with a static function which takes property file name as parameter and returns connection string.
    - Create a DBConnUtil class which holds static method which takes connection string as parameter file and returns connection object (Use method defined in DBPropertyUtil class to get the connection String).
  - **Main**
    - Create a class MainModule and demonstrate the functionalities in a menu driven application.

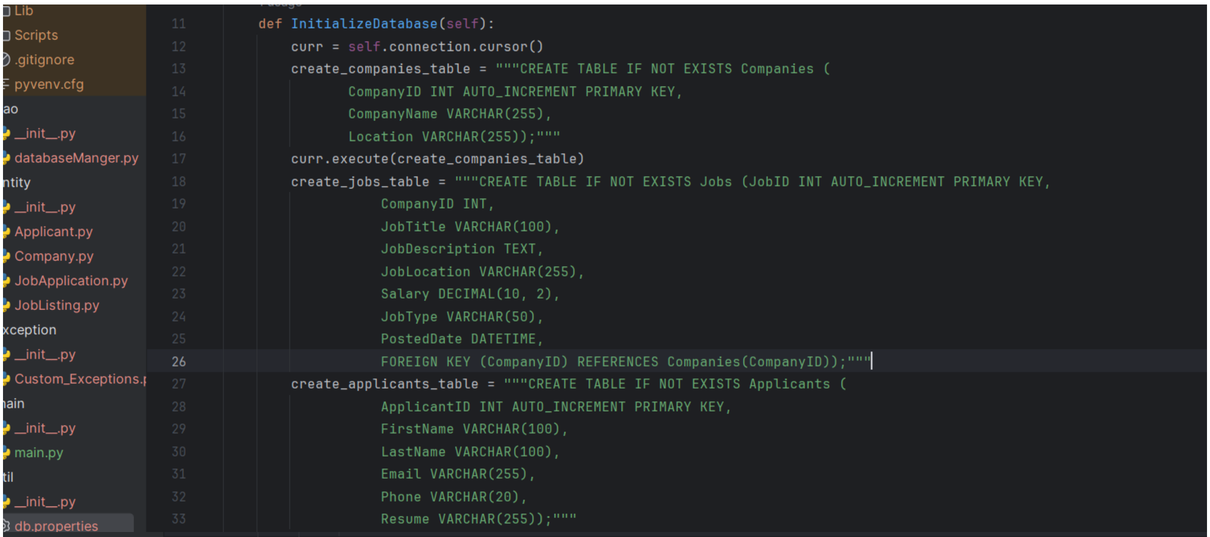


```
careerHub D:\Python\care
  .venv library root
    Lib
    Scripts
    .gitignore
    pyvenv.cfg
  dao
    __init__.py
    databaseManger.py
  entity
    __init__.py
    Applicant.py
    Company.py
    JobApplication.py
    JobListing.py
  exception
    __init__.py
    Custom_Exceptions.p
  main
    __init__.py
    main.py
  util
    __init__.py
    db.properties
    DBConnUtil.py
    DBPropertyUtil.py
```

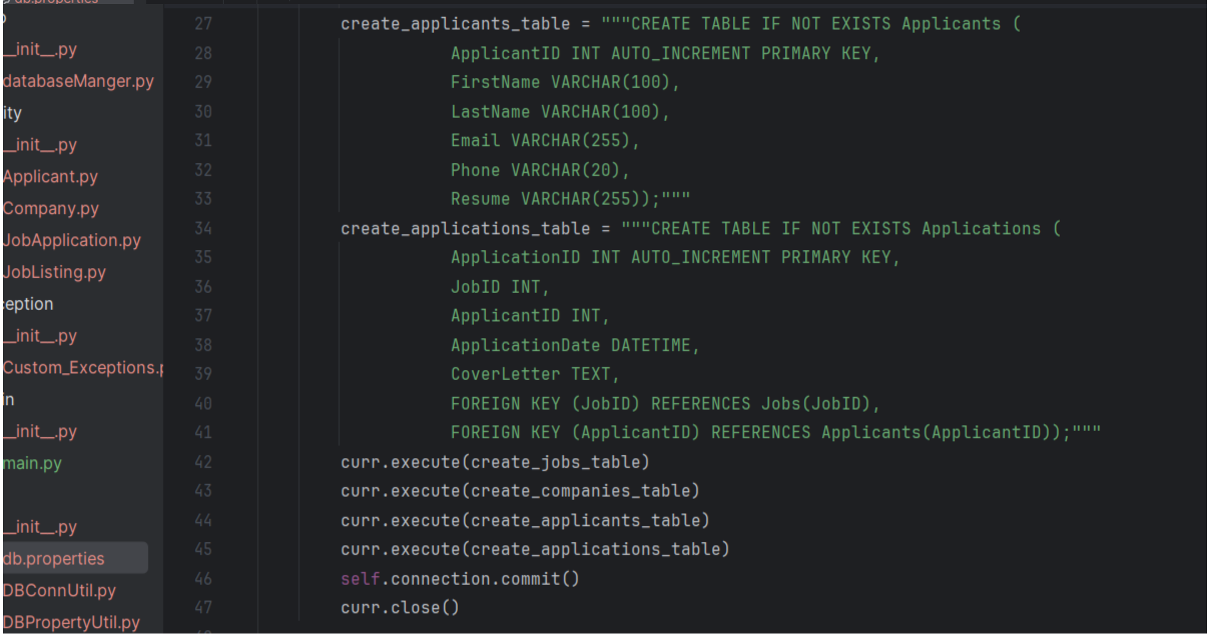
```python
1 usage   new *
def run(self):
    while True:
        self.display_menu()
        choice = input("Enter your choice: ")

        if choice == '1':
            try:
                self.db_manager.InitializeDatabase()
                print("Database initialized successfully!")
            except Exception as e:
                print("Error:", e)

        elif choice == '2':
            try:
                company_id = int(input("Enter Company ID: ")
                title = input("Enter Job Title: ")
                description = input("Enter Job Description:
                location = input("Enter Job Location: ")
                salary = float(input("Enter Salary: "))
                job_type = input("Enter Job Type: ")
                posted_date = datetime.now()
                job_listing = JobListing(company_id, title,
                self.db_manager.InsertJobListing(job_listing
                print("Job Listing inserted successfully!")
            except Exception as e:
                print("Error:", e)
```

(STRUCTURE)

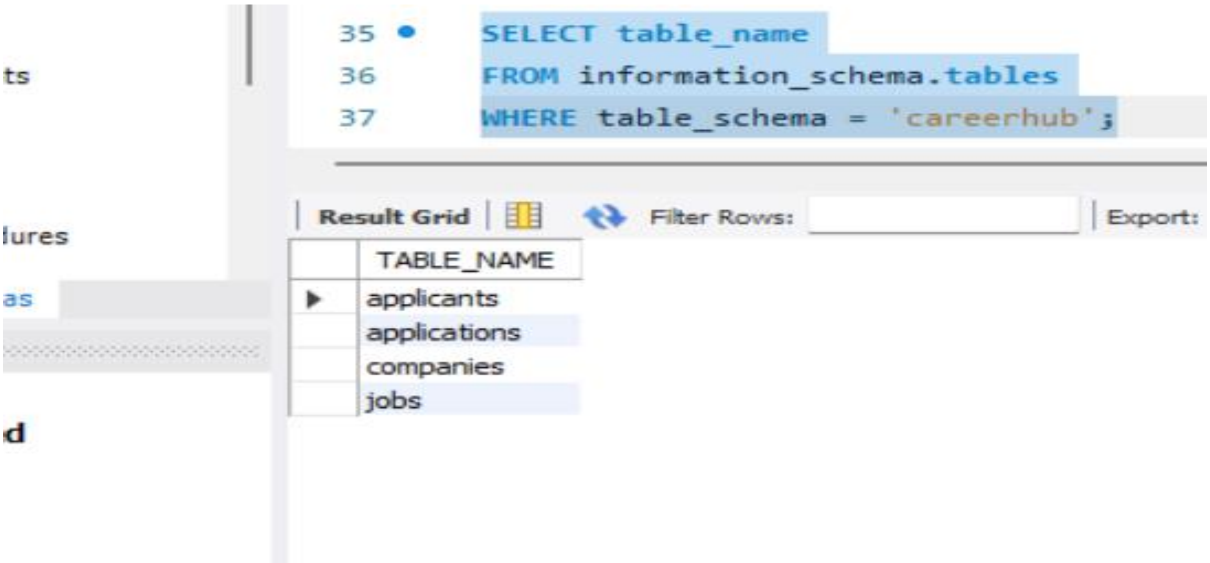
Create SQL Schema from the application, use the class attributes for table column names.



```
def InitializeDatabase(self):
    curr = self.connection.cursor()
    create_companies_table = """CREATE TABLE IF NOT EXISTS Companies (
           CompanyID INT AUTO_INCREMENT PRIMARY KEY,
           CompanyName VARCHAR(255),
           Location VARCHAR(255));"""
    curr.execute(create_companies_table)
    create_jobs_table = """CREATE TABLE IF NOT EXISTS Jobs (JobID INT AUTO_INCREMENT PRIMARY KEY,
               CompanyID INT,
               JobTitle VARCHAR(100),
               JobDescription TEXT,
               JobLocation VARCHAR(255),
               Salary DECIMAL(10, 2),
               JobType VARCHAR(50),
               PostedDate DATETIME,
               FOREIGN KEY (CompanyID) REFERENCES Companies(CompanyID));"""
    create_applicants_table = """CREATE TABLE IF NOT EXISTS Applicants (
               ApplicantID INT AUTO_INCREMENT PRIMARY KEY,
               FirstName VARCHAR(100),
               LastName VARCHAR(100),
               Email VARCHAR(255),
               Phone VARCHAR(20),
               Resume VARCHAR(255));"""
```



```
        create_applicants_table = """CREATE TABLE IF NOT EXISTS Applicants (
                   ApplicantID INT AUTO_INCREMENT PRIMARY KEY,
                   FirstName VARCHAR(100),
                   LastName VARCHAR(100),
                   Email VARCHAR(255),
                   Phone VARCHAR(20),
                   Resume VARCHAR(255));"""
        create_applications_table = """CREATE TABLE IF NOT EXISTS Applications (
                   ApplicationID INT AUTO_INCREMENT PRIMARY KEY,
                   JobID INT,
                   ApplicantID INT,
                   ApplicationDate DATETIME,
                   CoverLetter TEXT,
                   FOREIGN KEY (JobID) REFERENCES Jobs(JobID),
                   FOREIGN KEY (ApplicantID) REFERENCES Applicants(ApplicantID));"""
        curr.execute(create_jobs_table)
        curr.execute(create_companies_table)
        curr.execute(create_applicants_table)
        curr.execute(create_applications_table)
        self.connection.commit()
        curr.close()
```



```
SELECT table_name
FROM information_schema.tables
WHERE table_schema = 'careerhub';
```

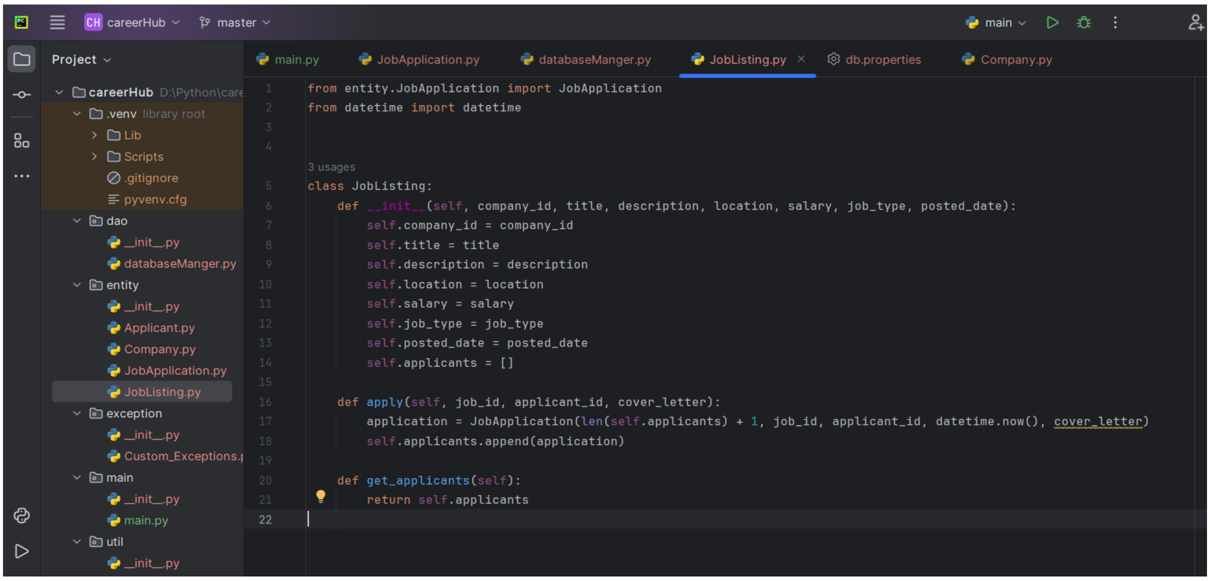
TABLE_NAME
applicants
applications
companies
jobs

Tables Created (MySQL workbench)

- JobID (int): A unique identifier for each job listing.
- CompanyID (int): A reference to the company offering the job.
- JobTitle (string): The title of the job.
- JobDescription (string): A detailed description of the job.
- JobLocation (string): The location of the job.
- Salary (decimal): The salary offered for the job.
- JobType (string): The type of job (e.g., Full-time, Part-time, Contract).
- PostedDate (DateTime): The date when the job was posted.

Methods:

- Apply(applicantID: int, coverLetter: string): Allows applicants to apply for the job by providing their ID and a cover letter.
- GetApplicants(): List<Applicant>: Retrieves a list of applicants who have applied for the job.



```python
from entity.JobApplication import JobApplication
from datetime import datetime


class JobListing:
    def __init__(self, company_id, title, description, location, salary, job_type, posted_date):
        self.company_id = company_id
        self.title = title
        self.description = description
        self.location = location
        self.salary = salary
        self.job_type = job_type
        self.posted_date = posted_date
        self.applicants = []

    def apply(self, job_id, applicant_id, cover_letter):
        application = JobApplication(len(self.applicants) + 1, job_id, applicant_id, datetime.now(), cover_letter)
        self.applicants.append(application)

    def get_applicants(self):
        return self.applicants

```

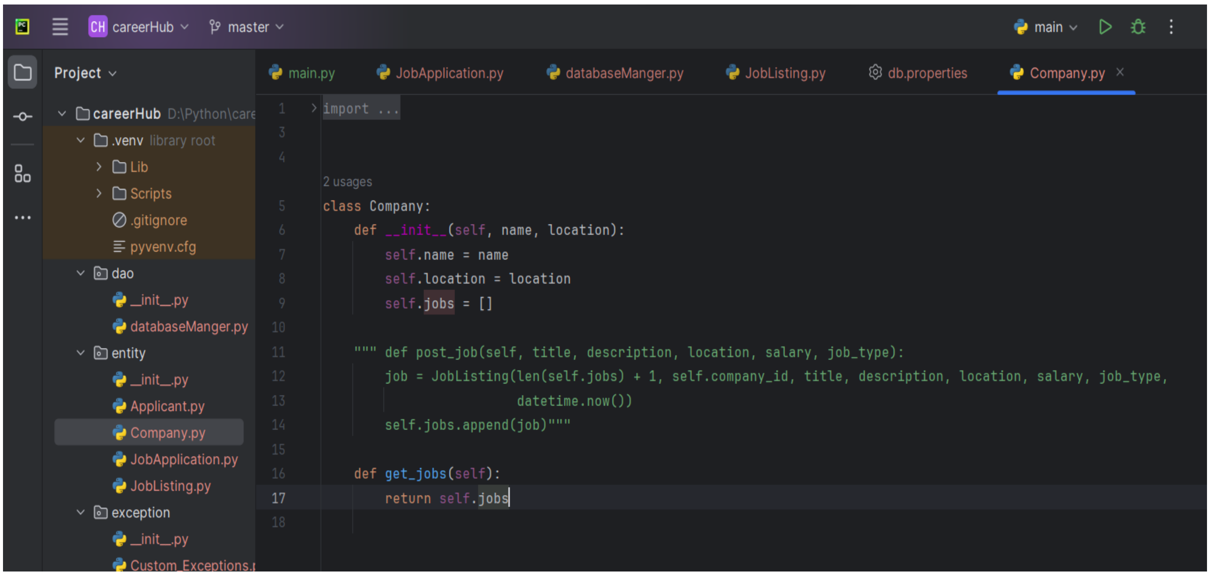
Company Class:

Attributes:

- CompanyID (int): A unique identifier for each company.
- CompanyName (string): The name of the hiring company.
- Location (string): The location of the company.

Methods:

- PostJob(jobTitle: string, jobDescription: string, jobLocation: string, salary: decimal, jobType: string): Allows a company to post a new job listing.
- GetJobs(): List<JobListing>: Retrieves a list of job listings posted by the company.



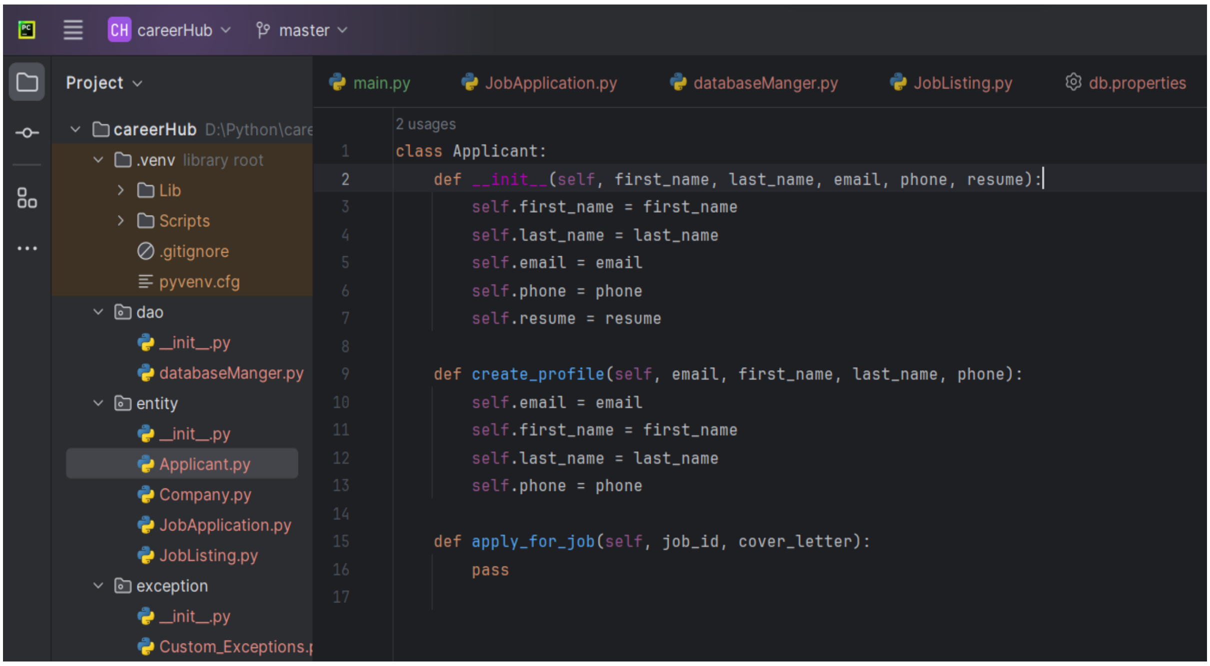
```python
import ...


class Company:
    def __init__(self, name, location):
        self.name = name
        self.location = location
        self.jobs = []

    """ def post_job(self, title, description, location, salary, job_type):
        job = JobListing(len(self.jobs) + 1, self.company_id, title, description, location, salary, job_type,
                         datetime.now())
        self.jobs.append(job)"""

    def get_jobs(self):
        return self.jobs

```

Applicant Class:
Attributes:

- ApplicantID (int): A unique identifier for each applicant.
- FirstName (string): The first name of the applicant.
- LastName (string): The last name of the applicant.
- Email (string): The email address of the applicant.
- Phone (string): The phone number of the applicant.
- Resume (string): The applicant's resume or a reference to the resume file.

Methods:

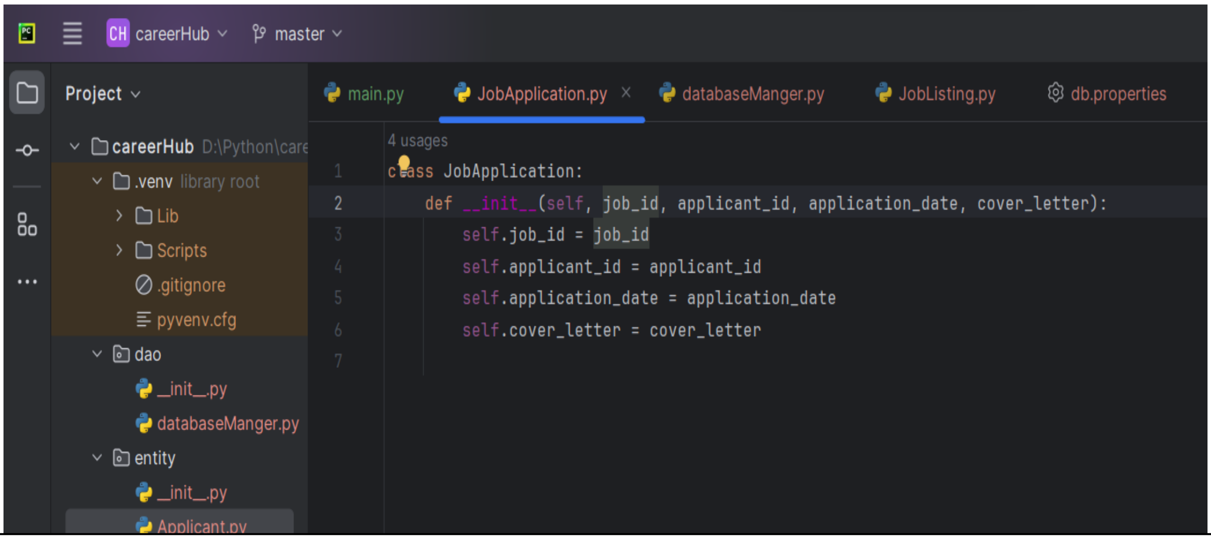
- CreateProfile(email: string, firstName: string, lastName: string, phone: string): Allows applicants to create a profile with their contact information.
- ApplyForJob(jobID: int, coverLetter: string): Enables applicants to apply for a specific job listing.



```python
class Applicant:
    def __init__(self, first_name, last_name, email, phone, resume):
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.phone = phone
        self.resume = resume

    def create_profile(self, email, first_name, last_name, phone):
        self.email = email
        self.first_name = first_name
        self.last_name = last_name
        self.phone = phone

    def apply_for_job(self, job_id, cover_letter):
        pass
```

JobApplication Class:
Attributes:

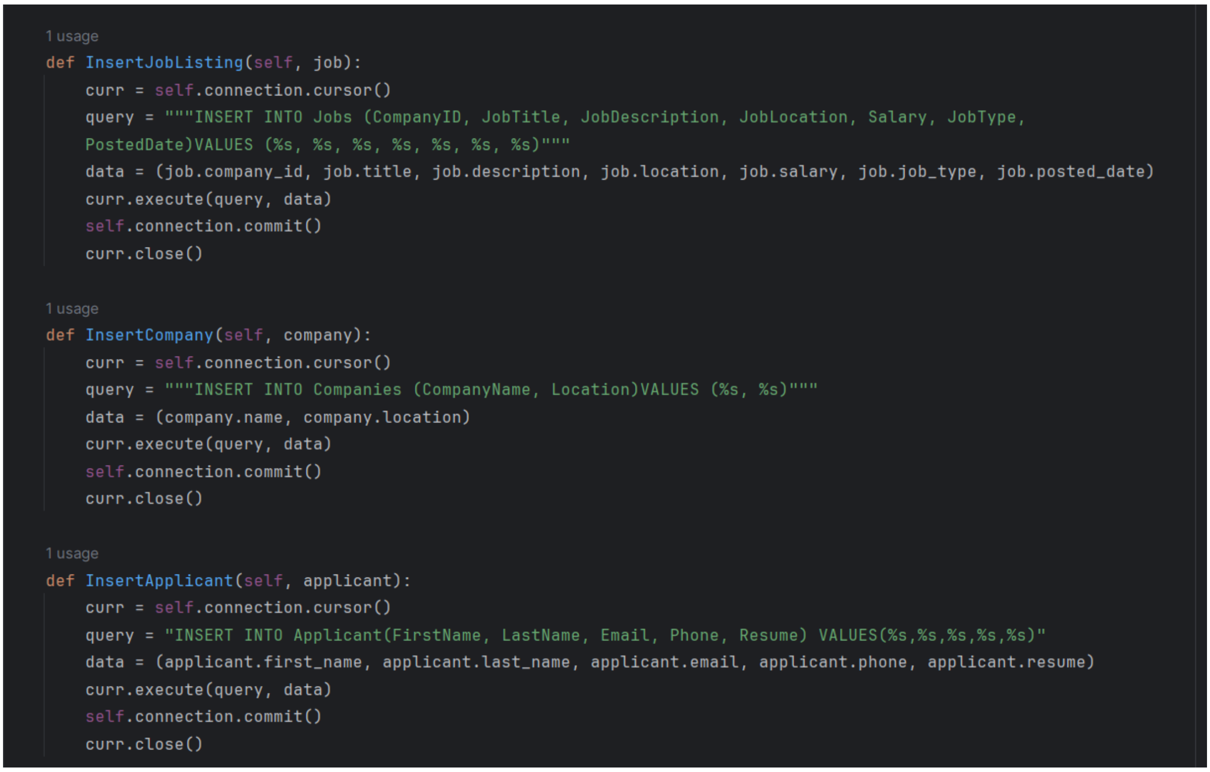
- ApplicationID (int): A unique identifier for each job application.
- JobID (int): A reference to the job listing.
- ApplicantID (int): A reference to the applicant.
- ApplicationDate (DateTime): The date and time when the application was submitted.
- CoverLetter (string): The cover letter submitted with the application.



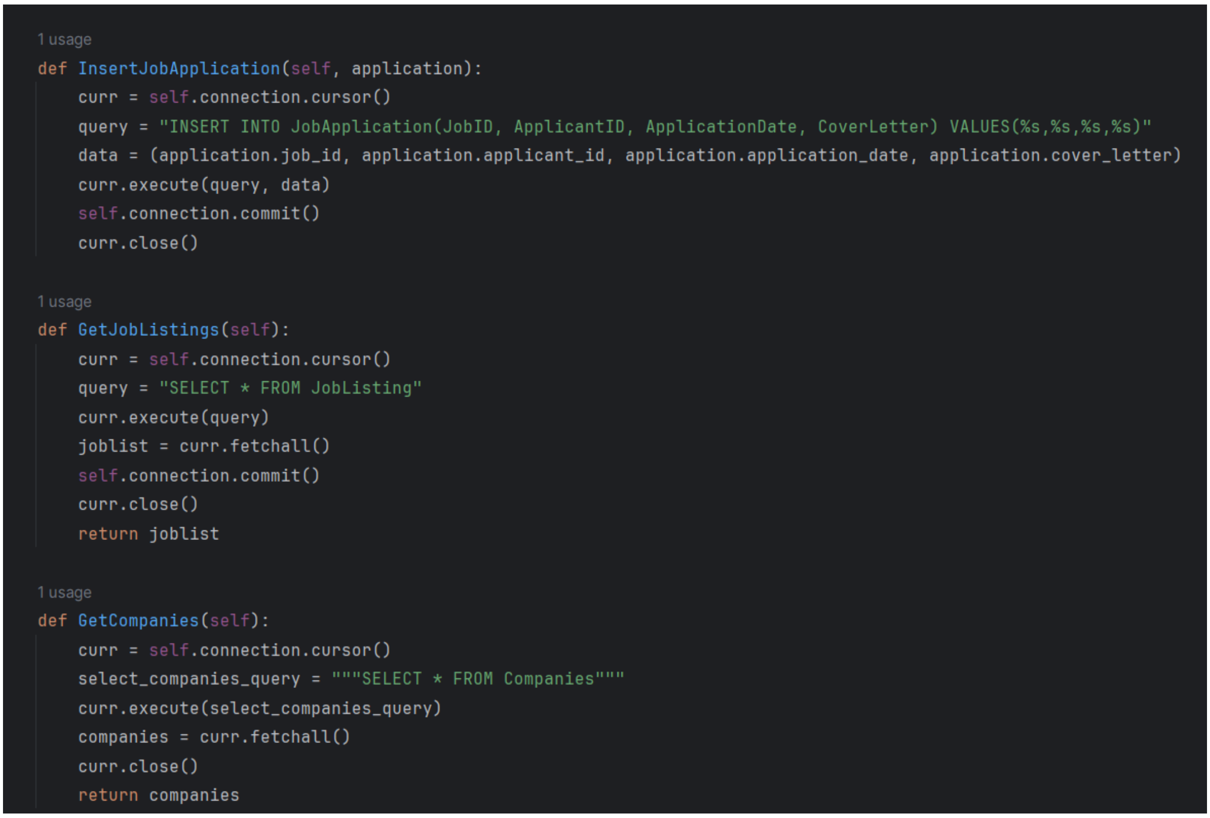
```python
class JobApplication:
    def __init__(self, job_id, applicant_id, application_date, cover_letter):
        self.job_id = job_id
        self.applicant_id = applicant_id
        self.application_date = application_date
        self.cover_letter = cover_letter
```

- InitializeDatabase(): Initializes the database schema and tables.
- InsertJobListing(job: JobListing): Inserts a new job listing into the "Jobs" table.
- InsertCompany(company: Company): Inserts a new company into the "Companies" table.
- InsertApplicant(applicant: Applicant): Inserts a new applicant into the "Applicants" table.
- InsertJobApplication(application: JobApplication): Inserts a new job application into the "Applications" table.
- GetJobListings(): List<JobListing>: Retrieves a list of all job listings.
- GetCompanies(): List<Company>: Retrieves a list of all companies.
- GetApplicants(): List<Applicant>: Retrieves a list of all applicants.
- GetApplicationsForJob(jobID: int): List<JobApplication>: Retrieves a list of job applications for a specific job listing.

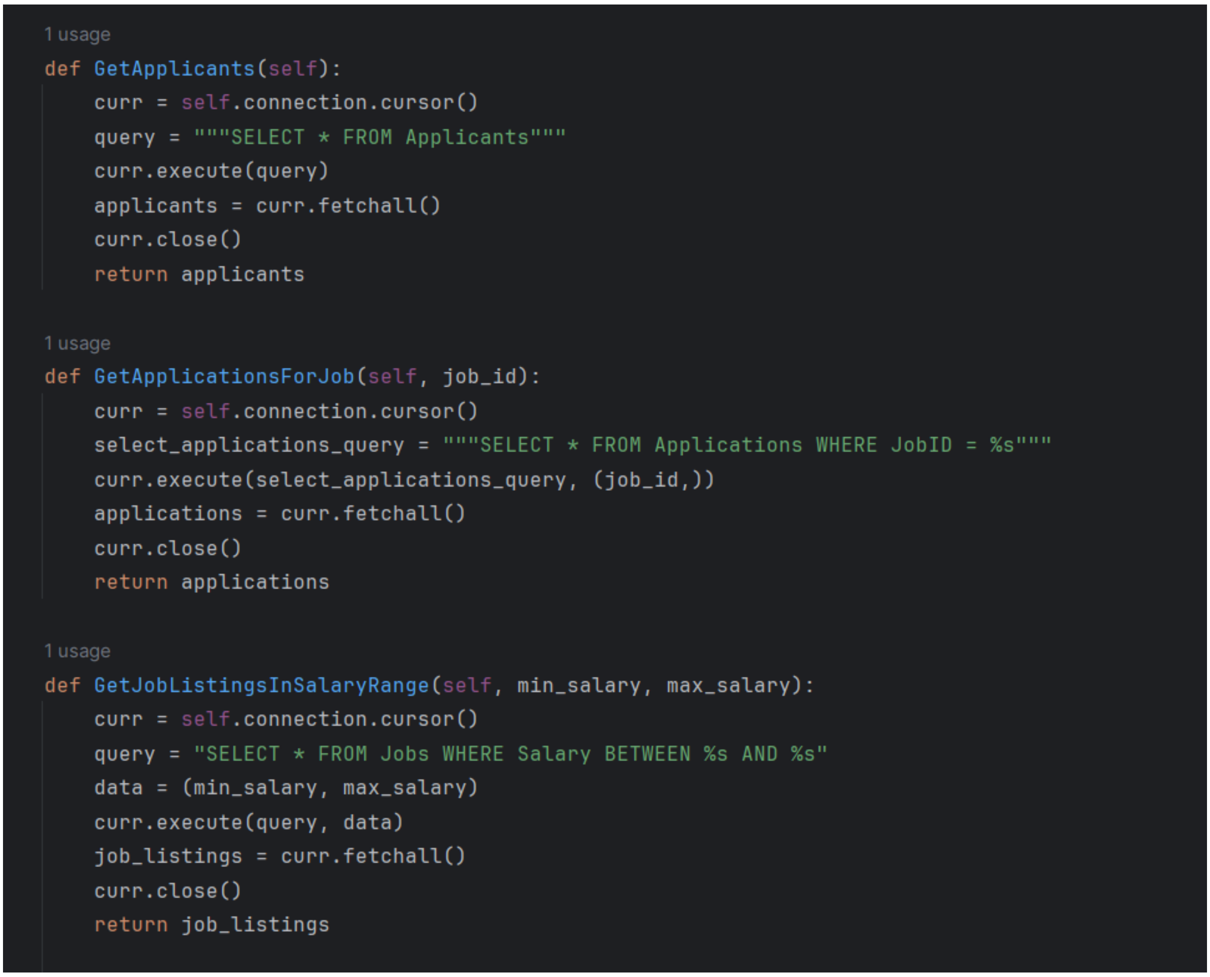
ALL THE METHODS ABOVE LISTED MEHTODS WITH THEIR IMPLEMENTATION



```
1 usage
def InsertJobListing(self, job):
    curr = self.connection.cursor()
    query = """INSERT INTO Jobs (CompanyID, JobTitle, JobDescription, JobLocation, Salary, JobType,
    PostedDate)VALUES (%s, %s, %s, %s, %s, %s, %s)"""
    data = (job.company_id, job.title, job.description, job.location, job.salary, job.job_type, job.posted_date)
    curr.execute(query, data)
    self.connection.commit()
    curr.close()

1 usage
def InsertCompany(self, company):
    curr = self.connection.cursor()
    query = """INSERT INTO Companies (CompanyName, Location)VALUES (%s, %s)"""
    data = (company.name, company.location)
    curr.execute(query, data)
    self.connection.commit()
    curr.close()

1 usage
def InsertApplicant(self, applicant):
    curr = self.connection.cursor()
    query = "INSERT INTO Applicant(FirstName, LastName, Email, Phone, Resume) VALUES(%s,%s,%s,%s,%s)"
    data = (applicant.first_name, applicant.last_name, applicant.email, applicant.phone, applicant.resume)
    curr.execute(query, data)
    self.connection.commit()
    curr.close()
```



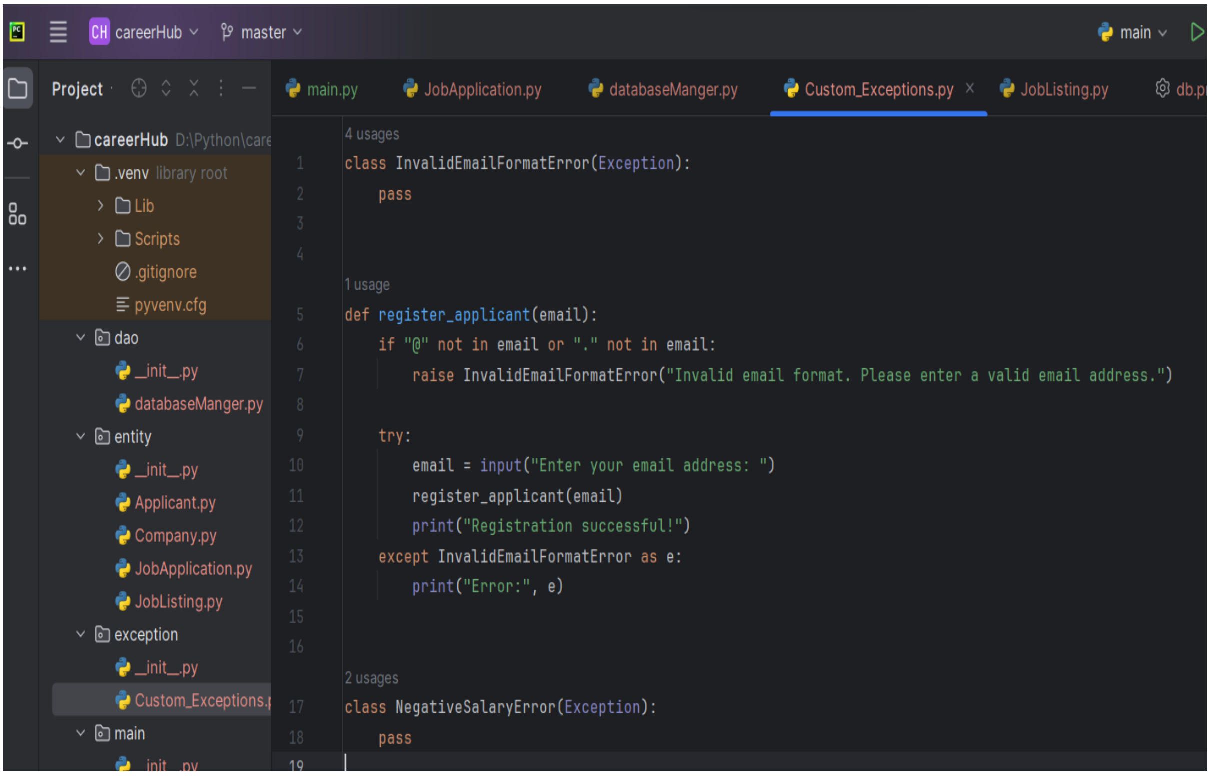
```
1 usage
def InsertJobApplication(self, application):
    curr = self.connection.cursor()
    query = "INSERT INTO JobApplication(JobID, ApplicantID, ApplicationDate, CoverLetter) VALUES(%s,%s,%s,%s)"
    data = (application.job_id, application.applicant_id, application.application_date, application.cover_letter)
    curr.execute(query, data)
    self.connection.commit()
    curr.close()

1 usage
def GetJobListings(self):
    curr = self.connection.cursor()
    query = "SELECT * FROM JobListing"
    curr.execute(query)
    joblist = curr.fetchall()
    self.connection.commit()
    curr.close()
    return joblist

1 usage
def GetCompanies(self):
    curr = self.connection.cursor()
    select_companies_query = """SELECT * FROM Companies"""
    curr.execute(select_companies_query)
    companies = curr.fetchall()
    curr.close()
    return companies
```



```
1 usage
def GetApplicants(self):
    curr = self.connection.cursor()
    query = """SELECT * FROM Applicants"""
    curr.execute(query)
    applicants = curr.fetchall()
    curr.close()
    return applicants

1 usage
def GetApplicationsForJob(self, job_id):
    curr = self.connection.cursor()
    select_applications_query = """SELECT * FROM Applications WHERE JobID = %s"""
    curr.execute(select_applications_query, (job_id,))
    applications = curr.fetchall()
    curr.close()
    return applications

1 usage
def GetJobListingsInSalaryRange(self, min_salary, max_salary):
    curr = self.connection.cursor()
    query = "SELECT * FROM Jobs WHERE Salary BETWEEN %s AND %s"
    data = (min_salary, max_salary)
    curr.execute(query, data)
    job_listings = curr.fetchall()
    curr.close()
    return job_listings
```

3.Exceptions handling
Create and implement the following exceptions in your application.

- Invalid Email Format Handling:
  - In the Job Board application, during the applicant registration process, users are required to enter their email addresses. Write a program that prompts the user to input an email address and implement exception handling to ensure that the email address follows a valid format (e.g., contains "@" and a valid domain). If the input is not valid, catch the exception and display an error message. If it is valid, proceed with registration.
- Salary Calculation Handling:
  - Create a program that calculates the average salary offered by companies for job listings. Implement exception handling to ensure that the salary values are non-negative when computing the average. If any salary is negative or invalid, catch the exception and display an error message, indicating the problematic job listings.
- File Upload Exception Handling:
  - In the Job Board application, applicants can upload their resumes as files. Write a program that handles file uploads and implements exception handling to catch and handle potential errors, such as file not found, file size exceeded, or file format not supported. Provide appropriate error messages in each case.
- Application Deadline Handling:
  - Develop a program that checks whether a job application is submitted before the application deadline. Implement exception handling to catch situations where an applicant tries to submit an application after the deadline has passed. Display a message indicating that the application is no longer accepted.
- Database Connection Handling:
  - In the Job Board application, database connectivity is crucial. Create a program that establishes a connection to the database to retrieve job listings. Implement exception handling to catch database-related exceptions, such as connection errors or SQL query errors. Display appropriate error messages and ensure graceful handling of these exceptions.

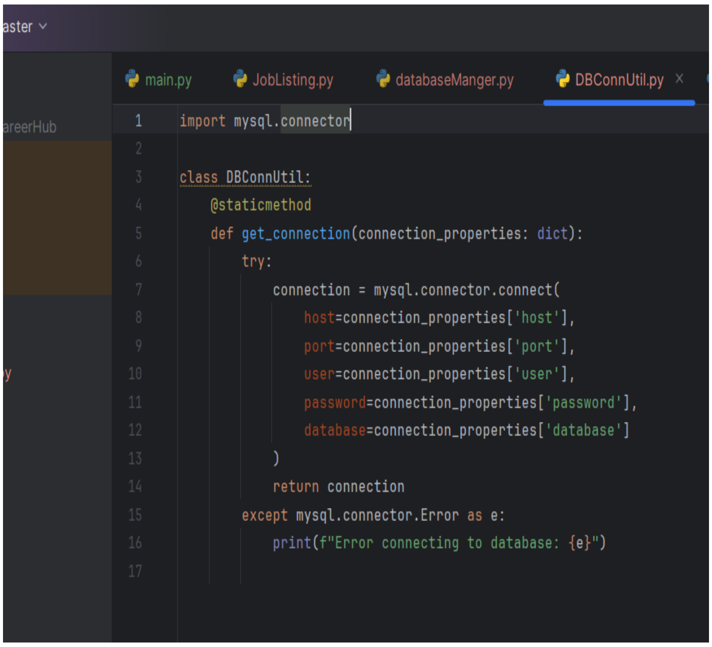


```
4 usages
class InvalidEmailFormatError(Exception):
    pass


1 usage
def register_applicant(email):
    if "@" not in email or "." not in email:
        raise InvalidEmailFormatError("Invalid email format. Please enter a valid email address.")

    try:
        email = input("Enter your email address: ")
        register_applicant(email)
        print("Registration successful!")
    except InvalidEmailFormatError as e:
        print("Error:", e)


2 usages
class NegativeSalaryError(Exception):
    pass
```

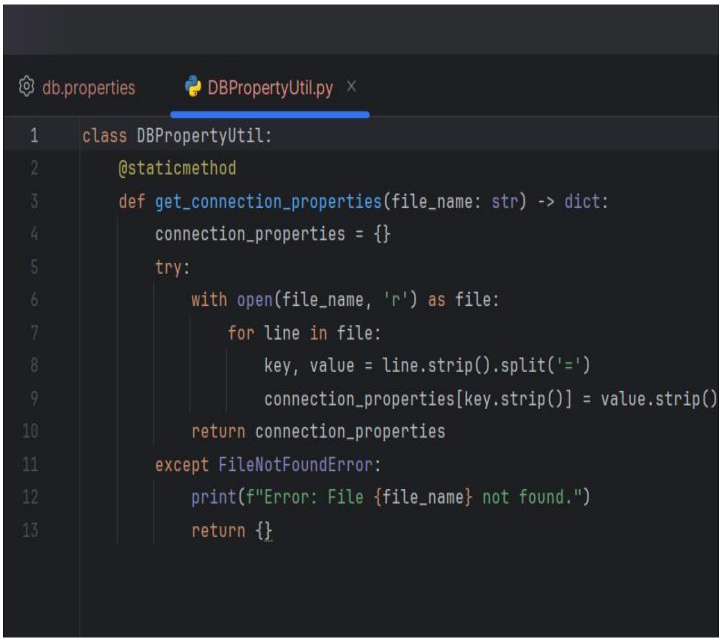
4.Database Connectivity

Create and implement the following tasks in your application.

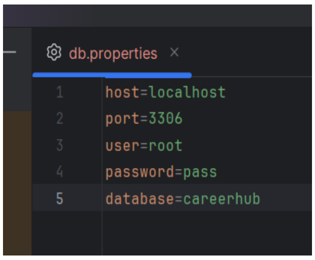
- **Job Listing Retrieval:** Write a program that connects to the database and retrieves all job listings from the "Jobs" table. Implement database connectivity using Entity Framework and display the job titles, company names, and salaries.
- **Applicant Profile Creation:** Create a program that allows applicants to create a profile by entering their information. Implement database connectivity to insert the applicant's data into the "Applicants" table. Handle potential database-related exceptions.
- **Job Application Submission:** Develop a program that allows applicants to apply for a specific job listing. Implement database connectivity to insert the job application details into the "Applications" table, including the applicant's ID and the job ID. Ensure that the program handles database connectivity and insertion exceptions.
- **Company Job Posting:** Write a program that enables companies to post new job listings. Implement database connectivity to insert job listings into the "Jobs" table, including the company's ID. Handle database-related exceptions and ensure the job posting is successful.
- **Salary Range Query:** Create a program that allows users to search for job listings within a specified salary range. Implement database connectivity to retrieve job listings that match the user's criteria, including job titles, company names, and salaries. Ensure the program handles database connectivity and query exceptions.



```python
import mysql.connector

class DBConnUtil:
    @staticmethod
    def get_connection(connection_properties: dict):
        try:
            connection = mysql.connector.connect(
                host=connection_properties['host'],
                port=connection_properties['port'],
                user=connection_properties['user'],
                password=connection_properties['password'],
                database=connection_properties['database']
            )
            return connection
        except mysql.connector.Error as e:
            print(f"Error connecting to database: {e}")
```



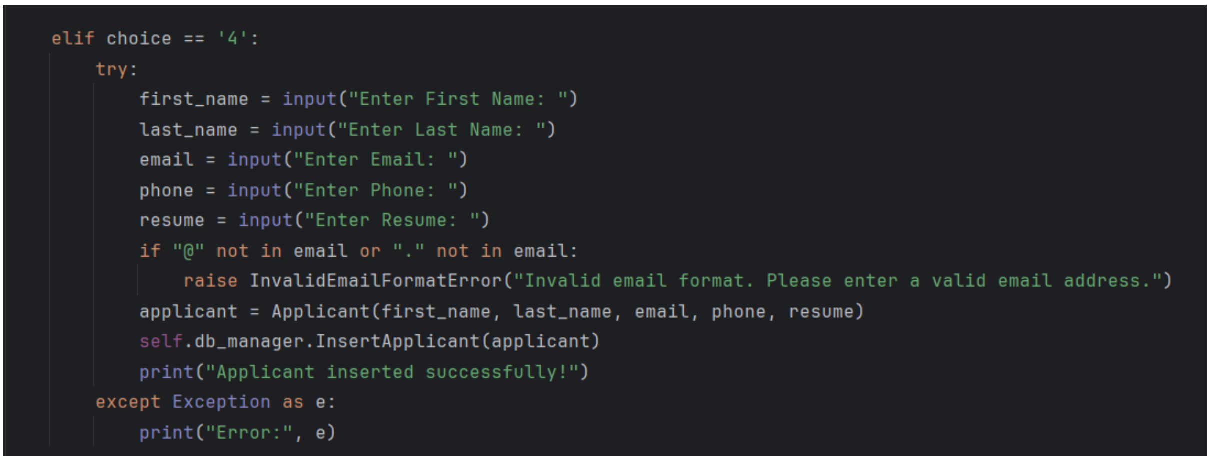
```python
class DBPropertyUtil:
    @staticmethod
    def get_connection_properties(file_name: str) -> dict:
        connection_properties = {}
        try:
            with open(file_name, 'r') as file:
                for line in file:
                    key, value = line.strip().split('=')
                    connection_properties[key.strip()] = value.strip()
            return connection_properties
        except FileNotFoundError:
            print(f"Error: File {file_name} not found.")
            return {}
```



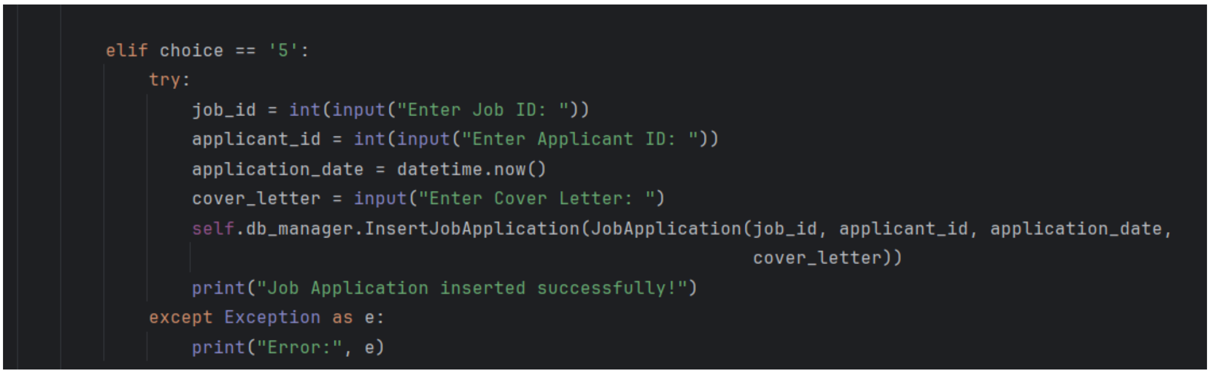
```
host=localhost
port=3306
user=root
password=pass
database=careerhub
```

(DAO package)

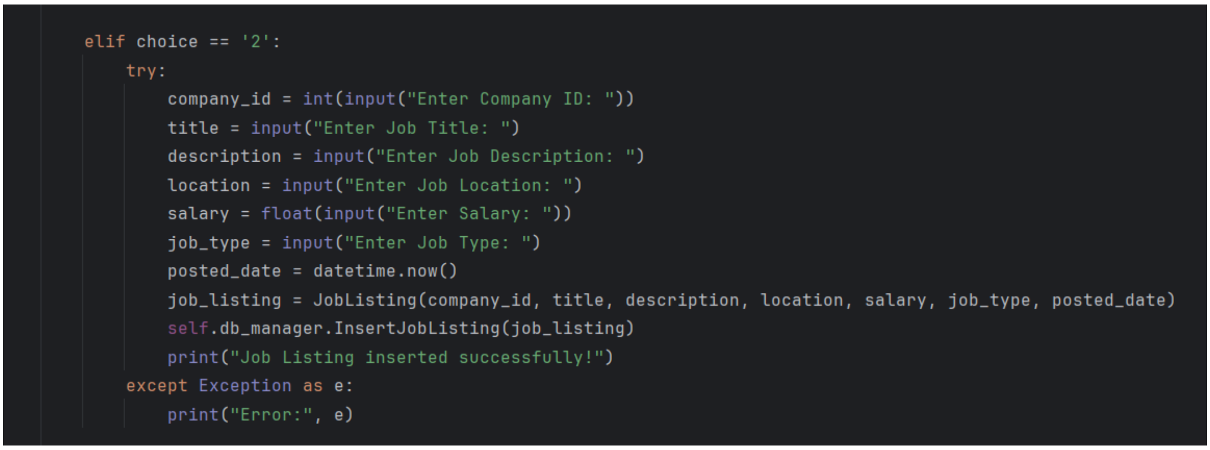
ALL THE OPERATION ARE CALLED FROM THE MAIN CLASS -



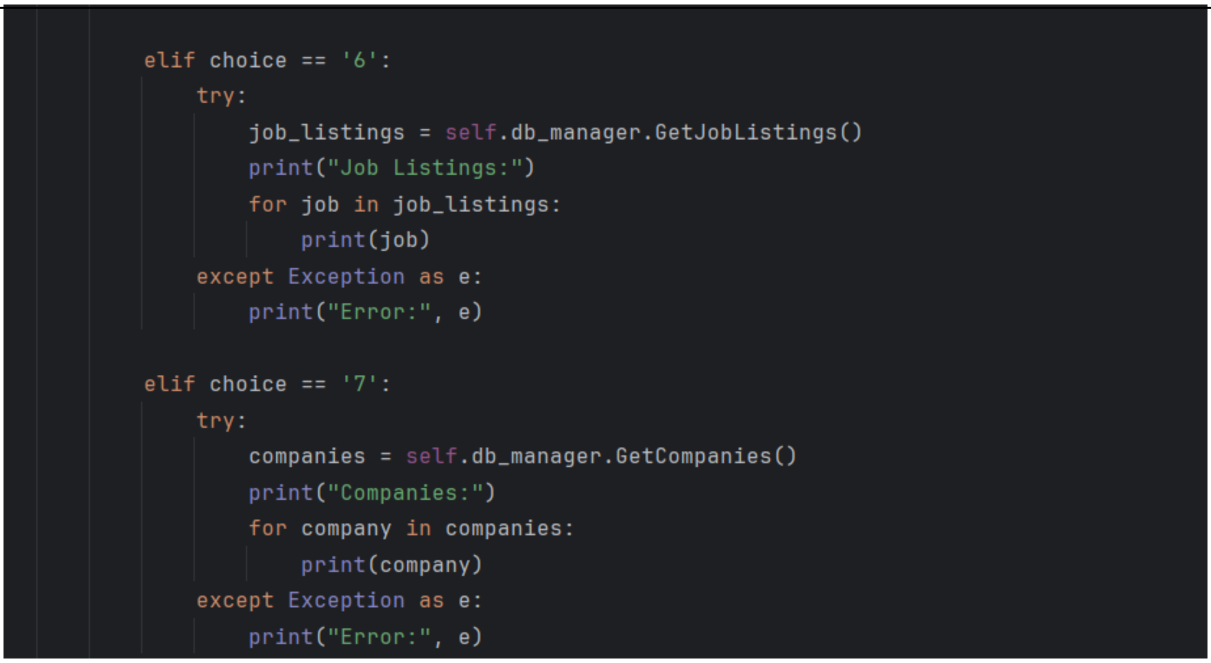
```python
elif choice == '4':
    try:
        first_name = input("Enter First Name: ")
        last_name = input("Enter Last Name: ")
        email = input("Enter Email: ")
        phone = input("Enter Phone: ")
        resume = input("Enter Resume: ")
        if "@" not in email or "." not in email:
            raise InvalidEmailFormatError("Invalid email format. Please enter a valid email address.")
        applicant = Applicant(first_name, last_name, email, phone, resume)
        self.db_manager.InsertApplicant(applicant)
        print("Applicant inserted successfully!")
    except Exception as e:
        print("Error:", e)
```



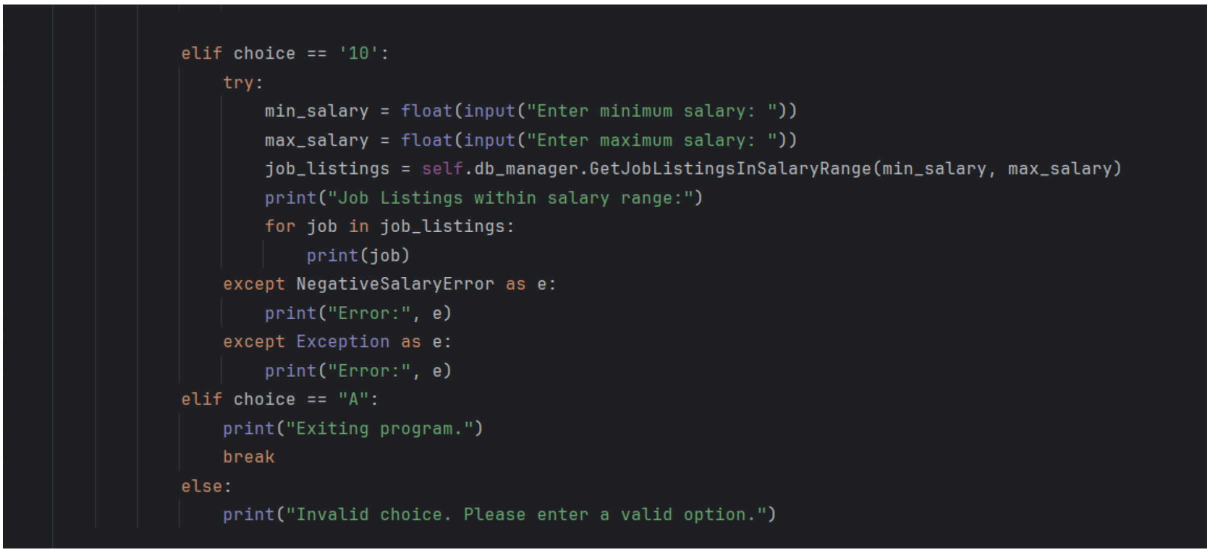
```python
elif choice == '5':
    try:
        job_id = int(input("Enter Job ID: "))
        applicant_id = int(input("Enter Applicant ID: "))
        application_date = datetime.now()
        cover_letter = input("Enter Cover Letter: ")
        self.db_manager.InsertJobApplication(JobApplication(job_id, applicant_id, application_date,
                                                            cover_letter))
        print("Job Application inserted successfully!")
    except Exception as e:
        print("Error:", e)
```



```python
elif choice == '2':
    try:
        company_id = int(input("Enter Company ID: "))
        title = input("Enter Job Title: ")
        description = input("Enter Job Description: ")
        location = input("Enter Job Location: ")
        salary = float(input("Enter Salary: "))
        job_type = input("Enter Job Type: ")
        posted_date = datetime.now()
        job_listing = JobListing(company_id, title, description, location, salary, job_type, posted_date)
        self.db_manager.InsertJobListing(job_listing)
        print("Job Listing inserted successfully!")
    except Exception as e:
        print("Error:", e)
```

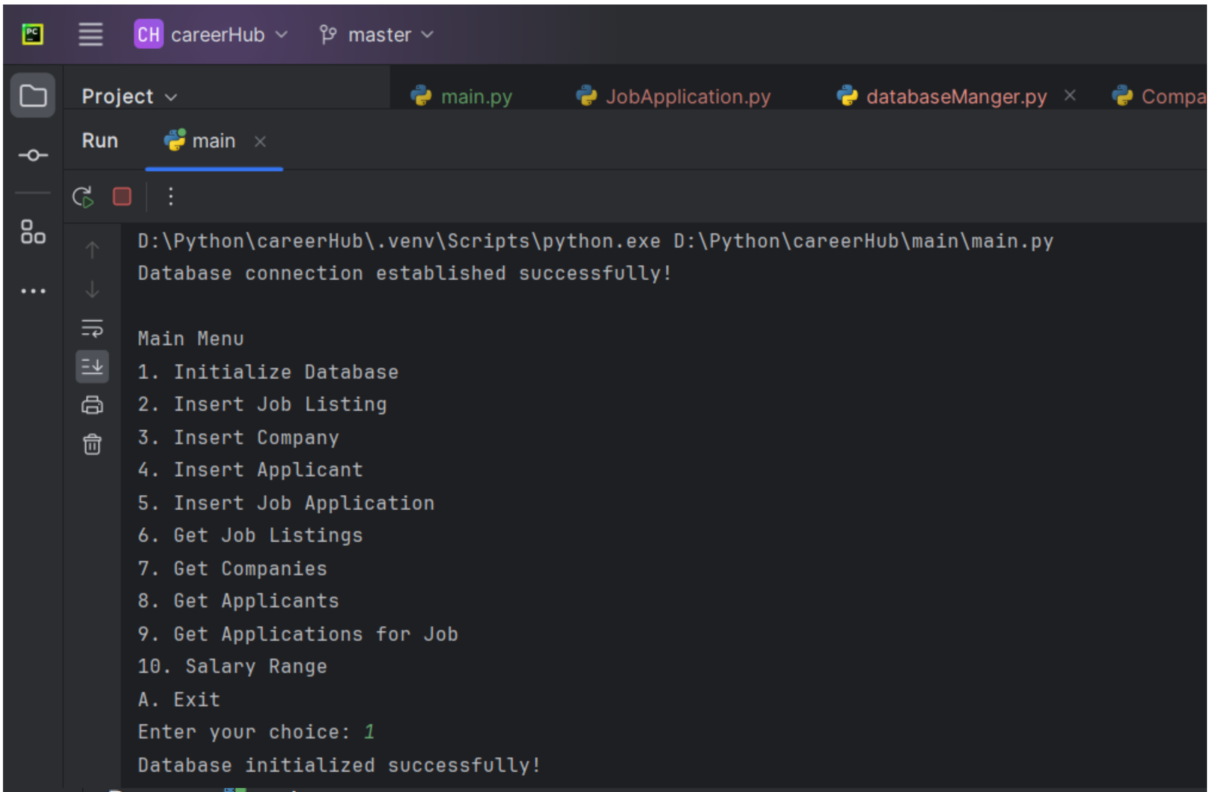


```python
            elif choice == '6':
                try:
                    job_listings = self.db_manager.GetJobListings()
                    print("Job Listings:")
                    for job in job_listings:
                        print(job)
                except Exception as e:
                    print("Error:", e)

            elif choice == '7':
                try:
                    companies = self.db_manager.GetCompanies()
                    print("Companies:")
                    for company in companies:
                        print(company)
                except Exception as e:
                    print("Error:", e)
```

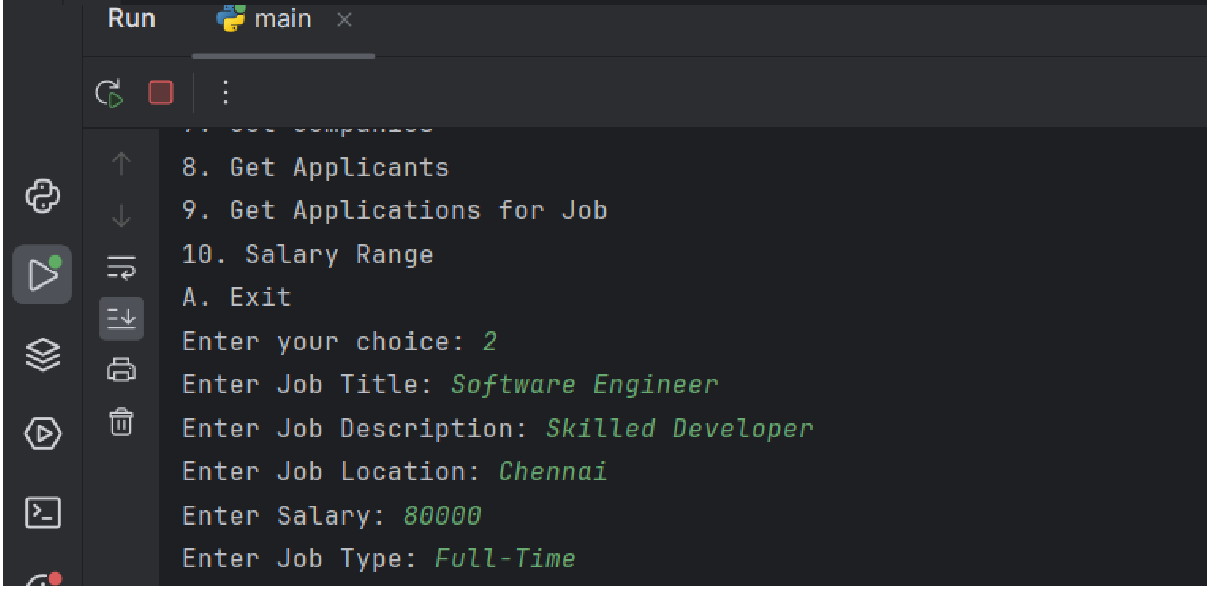


```python
            elif choice == '10':
                try:
                    min_salary = float(input("Enter minimum salary: "))
                    max_salary = float(input("Enter maximum salary: "))
                    job_listings = self.db_manager.GetJobListingsInSalaryRange(min_salary, max_salary)
                    print("Job Listings within salary range:")
                    for job in job_listings:
                        print(job)
                except NegativeSalaryError as e:
                    print("Error:", e)
                except Exception as e:
                    print("Error:", e)
            elif choice == "A":
                print("Exiting program.")
                break
            else:
                print("Invalid choice. Please enter a valid option.")
```

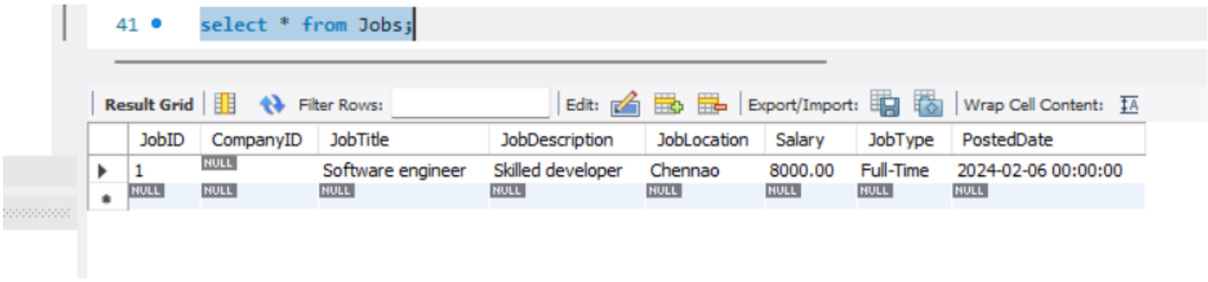
APPLICATION WORKING EXAMPLES



```
D:\Python\careerHub\.venv\Scripts\python.exe D:\Python\careerHub\main\main.py
Database connection established successfully!

Main Menu
1. Initialize Database
2. Insert Job Listing
3. Insert Company
4. Insert Applicant
5. Insert Job Application
6. Get Job Listings
7. Get Companies
8. Get Applicants
9. Get Applications for Job
10. Salary Range
A. Exit
Enter your choice: 1
Database initialized successfully!
```



```
8. Get Applicants
9. Get Applications for Job
10. Salary Range
A. Exit
Enter your choice: 2
Enter Job Title: Software Engineer
Enter Job Description: Skilled Developer
Enter Job Location: Chennai
Enter Salary: 80000
Enter Job Type: Full-Time
```



```
select * from Jobs;
```

JobID	CompanyID	JobTitle	JobDescription	JobLocation	Salary	JobType	PostedDate
1	NULL	Software engineer	Skilled developer	Chennao	8000.00	Full-Time	2024-02-06 00:00:00
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

---END---